Formative System Engineering Laboratory
Division of Synergetic Information Science
Graduate School of Information Science and Technology
Hokkaido University
http://kussharo.complex.eng.hokudai.ac.jp/~takty/

Takuto YAMAGIDA

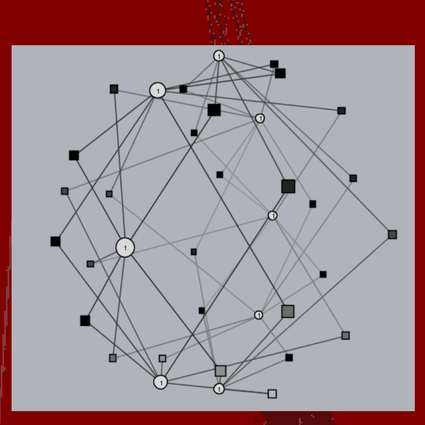
Coauthors: Masahito Kurihara, and Hidetoshi Nonaka

Fuzzy constraint satisfaction problems (FCSPs) are an extension of the classical (crisp) constraint satisfaction problem (CSP), which is a simple model for formulating problems in the real-world information systems studied in the field of artificial intelligence. FCSPs consist of variables, domains for the each variable, and constraints among the variables. Representing constraints by fuzzy relations, they provide suboptimal but useful solutions for a lot of problems which are hard to handle with the classical CSP.

Compelling FCSPs become and are used as a useful tool because a lot of general-purpose solvers are provided for FCSPs, once you formulate a problem as an FCSP, and then you can apply one of the solvers to it and obtain a solution. This formulation is however another problem because it is a process that entails highly abstracting the problem, it is not often determined uniquely, and it governs the efficiency of solving the problem. That is, the formulation is a problem left to humans, and can often be one of the research topics.

Since we have experiences of users of a tool FCSPs (in our previous work, we formulated the layout problem of GUIs as an FCSP), we consider that the situation of formulations improves by increasing the visibility of solvers' behavior in order to adjusting models developed once. When we utilized the framework of FCSPs for the problem, we were often confused because we were not able to comprehend behavior of solvers. Why does not a solver output a solution? Why does not it stop? What is the bottleneck of the model?

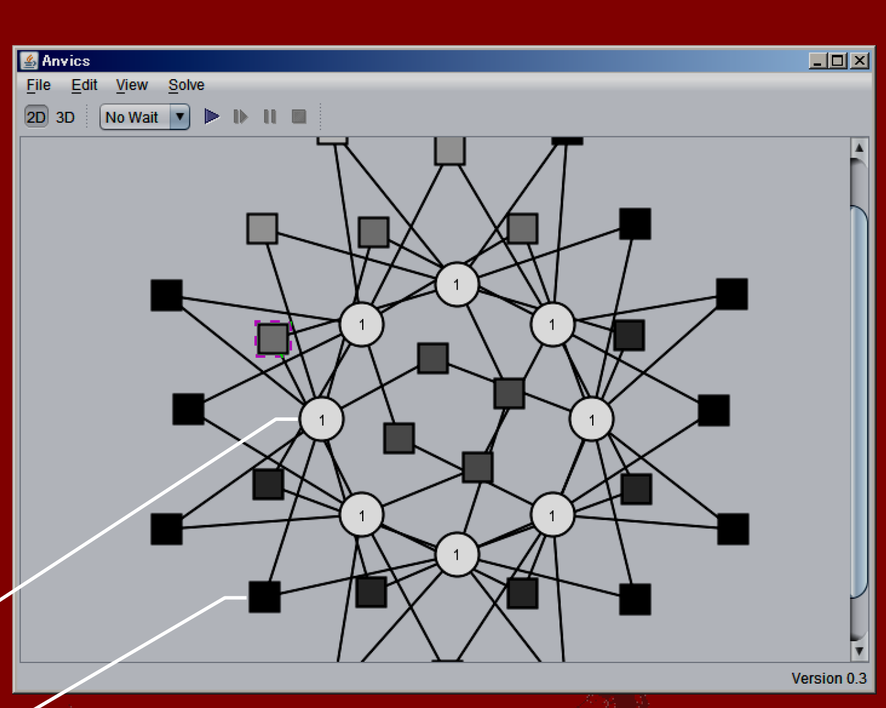
Picturing solving processes enables us feel the behavior of solvers. Here, we show our ongoing work for developing a tool for analyzing and visualizing FCSPs. This tool provides the two- and three-dimensional views of constraint graphs (a representation of FCSPs), and it realizes intuitive understanding of the behavior of solvers with sophisticated animations. In addition, the tool provides the functions of detecting endless loops of solvers, and debugging in similar way of integrated development environments.



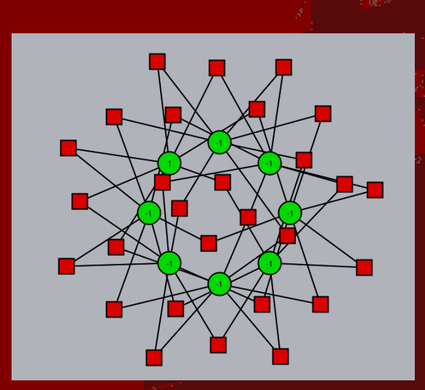
The 3D view of the same constraint graph as the right. Nodes are placed on the surface of a virtual sphere. Users can rotate by dragging.

A variable node. The value drawn here is the currently assigned value selected from its domain.

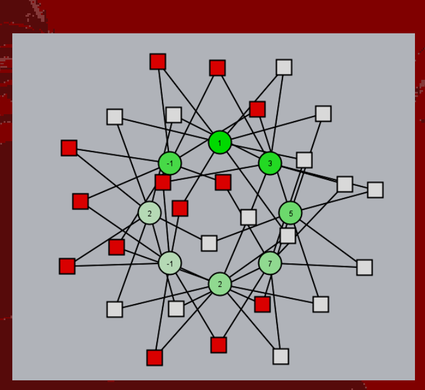
A constraint node. Its shading represents here the degree of satisfaction (black means the worst, whereas white means complete satisfaction).



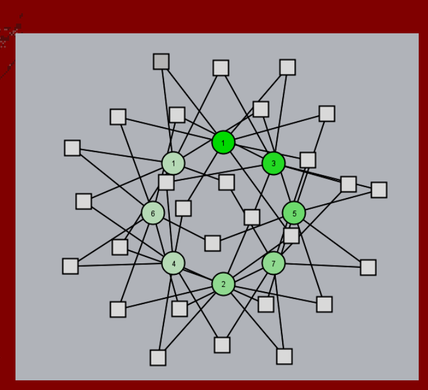

A screenshot of the tool we developed. In the center of the window, a constraint graph is drawn. Using the toolbar above, users can control the execution of solvers.



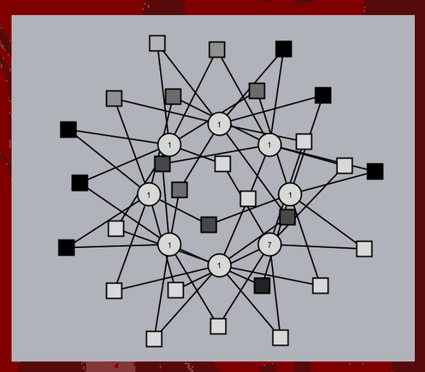
A-1: Before applying a systematic solver the forward checking (FC) to *8 queens on 8x7 board* problem. FC begins with the state that no values are assigned to the variables. In this figure, the constraint nodes are in red meaning satisfaction degrees are not defined for the empty variables.



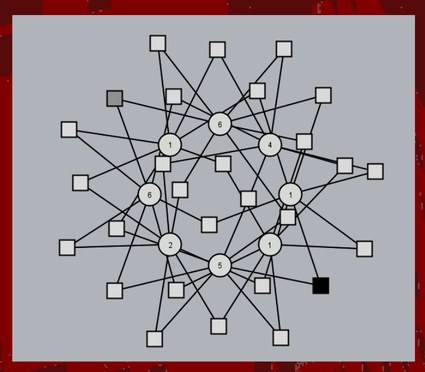
A-2: The FC tries to assign values of domains to the variables in turn from the topmost in a clockwise direction performing pruning of the domains. In this figure, the green density of the variable nodes represent the sizes of domains after pruned.



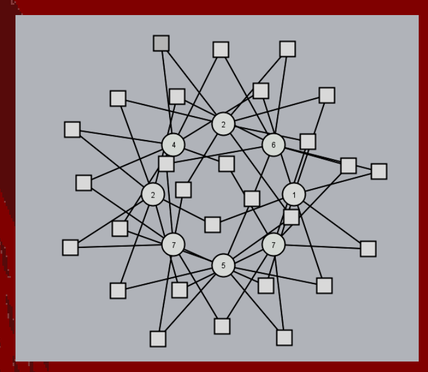
A-3: The FC finishes solving the problem. The domains are pruned more than figure A-2, and all constraints are almost satisfied. In this figure, values are assigned to all variables, and thus, there are no constraint nodes in red. The node in gray is the worst constraint.



B-1: Before applying a stochastic solver the breakout to the same problem as figure A. This solver begins with the state where values are assigned to all variables. Therefore, in this figure, there are no constraint nodes in red in comparison with figure A-1.



B-2: The breakout improves the worst constraints one by one iteratively. Every time trying to do, it puts a weight on a constraint in order to escape from local solutions and this is the reason of the name. By our tool, users can observe the solver as it is assigning at random.



B-3: The breakout finishes solving the problem. In the case of this example, this solver is faster than the FC. Since the solver selects one of the worst constraints stochastically to improve if they exist, obtained solutions might be different at execution.

# Feel the behavior of solvers